CS787: Advanced Algorithms	**Homework 2**
Out: 2/15/22	**Due: 2/24/22**

1 Game Tree evaluation

In class we saw a randomized algorithm for game tree evaluation in complete binary trees with n leaves where players alternate moves, that evaluated $O(n^{0.793})$ leaves in expectation. We considered the case where leaves have value 0 or 1 indicating whether the first player wins or loses. More generally, we can have arbitrary values in the leaves and player 1 tries to maximize the value while player 2 tries to minimize it.

a. Show that any deterministic algorithm must query the values at all leaves by expanding on the proof sketch we did in class.

b. With more general values, when evaluating one subtree of a node it is less clear whether we need to evaluate the other one as well. Devise a strategy that allows to prune the search space efficiently that still requires at most $O(n^{0.793})$ leaf evaluations.
Hint: Reduce to the case with 0 or 1 values. For every node compute the value of one subtree, then check if the other subtree has better value and do a full evaluation only when it does.

2 Game Tree evaluation with noisy leaves

Consider a generalization of the binary Game Tree we saw in class. The tree still has n leaves but may not be binary as nodes may have an arbitrary number of children. Evaluating such a tree requires at most n evaluations of the leaves to figure out whether the first player wins or not. In this variant, checking the value of a leaf may not be yield the same value every time it is queried. In fact, with probability $1/3$ the outcome of the leaf will be 0 and only with probability $2/3$ it will be the correct value. That is, if the true value is 0, the answer will always be 0 but if the true answer is 1, we will observe 1 with probability $2/3$.

a. Show that $O(n \log n)$ evaluations suffice to compute the true value of the game tree with probability at least $1 - 1/n$.

b. Show that $O(n)$ evaluations suffice to compute the true value of the game tree with probability at least 99%.
Hint: Consider querying a leaf repeatedly until the observed value is 1. The expected number of evaluations is 1.5 if the true value is 1, and ∞ if the true value is 0. Devise an evaluation strategy guaranteeing that for every subtree with k leaves the expected number of evaluations to see a 1 is $1.5k$ if the true value is 1 and ∞ if the true value is 0.

c. [Optional] Suppose that with probability $1/3$ the reported value is not 0 but arbitrary. Show that $O(n)$ evaluations suffice to compute the true value of the game tree with probability at least 99%.

3 Perfect and Maximum Matchings in non-bipartite graphs

A matching in a general undirected graph is a pairing of the n vertices of the graph, such that every vertex appears in at most one pair and every pair of vertices share an edge. A maximum matching is a valid matching where the number of pairs is maximized while a perfect matching has exactly $n/2$ pairs. Define the Tutte matrix A of the graph with entries given by variables $x_{u,v}$ for $u < v$ as:

$$A_{u,v} = \begin{cases} x_{u,v} & \text{if } (u,v) \in E \text{ and } u < v \\ -x_{v,u} & \text{if } (u,v) \in E \text{ and } u > v \\ 0 & \text{otherwise} \end{cases} .$$

i.e. for any i, j, $A_{i,j} = -A_{j,i}$.

a. [Optional] Show that the determinant of A is not identically equal to the zero polynomial if and only if there is a perfect matching in the graph.

b. Devise a randomized algorithm for testing if a graph has a perfect matching. Analyze its running time and success probability.

c. Show that the algorithm can be extended to compute the perfect matching if one exists.

d. Show how to extend the algorithm to compute a maximum matching.
Hint: Consider first the problem of checking whether there is a matching of size at least k.

4 Reachability in Temporal Graphs

Consider an empty graph G with n nodes but no edges. At every time step $t = 1...T$, an (undirected) edge appears between two nodes u_t and v_t but is only available for that time step. Your goal is to compute for any pair of nodes u, v whether it is possible to go from u to v. For example, consider the case with 3 nodes and edges $(1, 2)$ and $(2, 3)$ appearing at time steps 1 and 2, respectively. It is possible to go from node 1 to 3 by visiting node 2 at time step 1 and node 3 at time step 2 but it is not possible to go from node 3 to 1.

a. Let A be a matrix, where $A_{u,v}$ denotes whether u is reachable from v. Initially, $A_{uv} = 1$ if $u = v$ and 0 otherwise. How does the matrix change when an (undirected) edge (u, v) appears?

b. For any u, let $p_u(x; a, b) = \sum_{v=a}^{b} A_{u,v} x^v$ be a polynomial of degree at most b. For a fixed x, design a data structure that allows querying $p_u(x; a, b)$ in time $O(\log n)$ and allows updates in time $O(k \log n)$ when k entries of A change.
Hint: You may use a prefix sum data-structure like Binary trees or Fenwick trees

c. Show that using the above data structure, it is possible to find all k entries in A that become 1 when a new edge (u, v) appears with high probability in time $O(k \log^2 n)$.
Hint: Use polynomial identity testing and binary search. If at least one entry changed it must be that $p_u(x; 1, n) \neq p_v(x; 1, n)$ with high probability over a random x.

d. Conclude with an algorithm that achieves runtime $O(T \log n + n^2 \log^2 n)$.

You may assume that all polynomial computations are done modulo a large enough prime P.